On the Equivalence of Controllability and the Input Output Conformance Testing Relation

A. Khan P. Falkman M. Fabian

Chalmers University of Technology,
Department of Electrical Engineering, 41296 Göteborg, Sweden
{*kadnan, petter.falkman, fabian*}*@chalmers.se*

Abstract: In this paper, the relation between controllability and the IOCO testing relation is examined. Based on a natural and common notion of controllability, where uncontrollable events are interpreted as outputs from the plant, and viewing an implementation under test as a plant, the IOCO testing relation is equivalent to controllability. Further, it is shown how supervisor synthesis can be used to algorithmically make an implementation IOCO with respect to its specification. This can be done either by restricting the implementation to the supremal controllable sublanguage, or extending the specification to the infimal controllable superlanguage, of the implementation and the specification. Both alternatives seem to be equally viable, and the choice between them seem strongly application dependent.

Keywords: Input-Output Conformance, Controllablility, Supervisory Control Theory, Model-Based Testing, Discrete Event Systems

1. INTRODUCTION

Many types of engineered systems, such as manufacturing systems, traffic systems etc, are driven by software and are getting more and more automated for the benefit of society. In manufacturing systems, increased number of automated machines and resources have resulted in better production rates and quality of products. In automotive systems, self-driving cars promise to bring about beneficial changes to our environment, like less traffic, and less accidents.

However, the control programs that implement the functionality of these systems are complex and hard to develop so that the resulting system behavior is logically correct. In addition, in many cases these systems are safety-critical, meaning that failures can have catastrophic outcomes and must be avoided. Typically implementation is manual, and based on design documents that are informal in nature, mainly expressed in natural language, which further emphasizes the problem.

Many of these systems, and especially their decision making logic, can be modeled as *discrete event systems*, which are systems that evolve dynamically on the occurrence of events, while at each time instant occupying a specific state where certain conditions hold. Using this modeling formalism, many methods have been developed to help implementing correct behaviors from requirements specifications. This paper deals with two *formal* such methods, the *supervisory control theory* (Ramadge and Wonham, 1987), which takes a "correct-by-construction" approach, and *model-based testing* (Utting and Legeard, 2007), which applies testing techniques to a model of an implementation.

The supervisory control theory (Ramadge and Wonham, 1987) allows the automatic generation of controllers known as *supervisors* based on a model of the uncontrolled system, the *plant*, and the provided specifications. This process of automatic generation of supervisors is called *synthesis*. Since a controller only has partial control over the events that the plant generates, some of the events are *uncontrollable*, for the specification to be implementable on the plant, the property of *controllability* has to be satisfied. If the specification is not controllable with respect to the plant, the specification is automatically modified by the synthesis procedure, resulting in the supervisor which is guaranteed to be controllable with respect to the plant.

Model-based testing (Utting and Legeard, 2007) is a formal approach to subject a model of an implementation to series of *tests* that try to falsify the specification according to which the implementation was created, in order to find faults in the implementation. To formalize this, the concept of *input-output conformance* (IOCO) was proposed by Tretmans (1996). Similar to the supervisory control theory, the IOCO testing relation depends on two models, a specification model and an implementation model. In the IOCO testing relation, the specification provides the basis for the behavior of the implementation, in that it dynamically defines the outputs that the implementation is allowed to emit. If other than the specified outputs are emitted by the implementation, it is not IOCO with respect to the specification and must be modified. The testing literature, does not (to the best of the author's knowledge) describe any approach that modifies the implementation automatically.

⋆ This work has been carried out at the Wingquist Laboratory VINN Excellence Centre within the Production Area of Advance at Chalmers. It has been supported by Vinnova FFI VIRTCOM (2014-01408), and ITEA3 Vinnova ENTOC (2016-02716).

1.1 Contribution

In this paper, the equivalence between controllability and the IOCO testing relation is detailed. This equivalence is established in terms of viewing the implementation model as the plant, the uncontrollable events as outputs from the plant, and a semantic comparison between the respective formal definitions of controllability and IOCO. It is further shown how synthesis, as defined within the SCT, can be used to algorithmically make an implementation IOCO with respect to its specification, either by restricting the implementation or extending the specification.

1.2 Outline

This paper is structured as follows. In Section 2 background regarding the SCT is given. In Section 3, the IOCO testing relation is introduced. In Section 4, the equivalence between controllability and IOCO is established with the help of an example. Section 5 introduces a remedy for non-IOCO systems based on supervisor synthesis. Finally, Section 6 concludes the paper and presents some future work directions.

2. SUPERVISORY CONTROL THEORY

The Supervisory Control Theory (SCT) was introduced by Ramadge and Wonham (1987) to take a control-theoretic approach on discrete event systems. The main idea behind this theory is the automatic synthesis of controllers, called *supervisors*, for a system under scrutiny. The essential elements required to synthesize a supervisor are *specification* and *plant* models.

The plant is a model of all the possible behavior of the uncontrolled system, while the specification models the desired controlled behavior. The task of synthesis is now to automatically calculate a supervisor, given the plant and the specification, such that when this supervisor controls the plant the closed-loop system exhibits a behavior that is guaranteed to remain within the specification, while always being able to complete some desired task.

The supervisor effects its control on the plant by dynamically disabling events from occurring. Thus, there is an asymmetric feedback-loop between the plant and the supervisor, see Fig 1 (left), the plant generates events, while the supervisor at each global state disables a subset of the events to guarantee that the closed-loop system never goes outside the specified behavior. However, not all events are subject to disablement by the supervisor; *controllable* events can be disabled, while *uncontrollable* events cannot. This must be considered when synthesizing the supervisor; the synthesis procedure must produce a supervisor that is *controllable* with respect to the plant and the uncontrollable events.

In addition, some global states are of significant interest, and are therefore *marked* by the specification. At least one of these marked states must always be reachable from any state in the closed-loop system,. This poses a requirement on the supervisor to be *non-blocking*.

Furthermore, it is common to require that the supervisor should disable events only when enabling such events

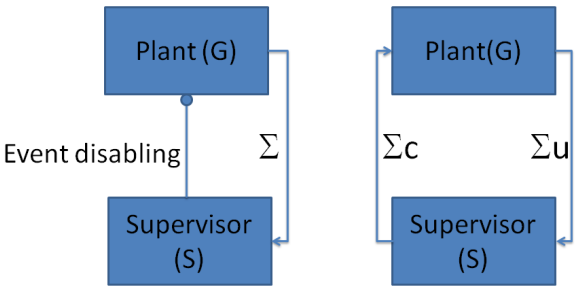

Fig. 1. Asymmetric (left) and symmetric (right) supervisor feedback loops.

would inevitably lead to violating the controllability or non-blocking properties. This is captured by the requirement of the supervisor to be *minimally restrictive* [1] It is known that a minimally restrictive, controllable and non-blocking supervisor always exists, is computable in the regular case, and is unique (Ramadge and Wonham, 1987; Cassandras and Lafortune, 2009).

Though Ramadge and Wonham (1987) originally took a formal language-theoretic viewpoint, for calculation purposes it is beneficial to model the plant, the specification, and the supervisor as finite-state machines (Cassandras and Lafortune, 2009).

Definition 1. A *finite-state machine* (FSM) A is a 5-tuple, $A = \langle Q_A, \Sigma_A, i_A, \delta_A, M_A \rangle$ where

- Q_A is a finite non-empty set of states;
- Σ_A is a finite non-empty set of events, the *alphabet*;
- i_A is the initial state, $i_A \in Q_A$;
- $\delta_A : Q_A \times \Sigma_A \to Q_A$ is the transition function;
- M_A is the set of marked states, $M_A \subseteq Q_A$.

The transition function δ_A is a partial function not necessarily defined for all combinations of states and events. An event $\sigma \in \Sigma_A$ is said to be *enabled* in a state $q \in Q_A$ if $\delta(\sigma, q)$ is defined. Also, δ_A can be extended to elements of Σ_A^*, the Kleene closure of Σ_A, which are sequences of events, in such a way that for $s\sigma \in \Sigma_A^*$ and $q \in Q_A$, $\delta_A(s\sigma, q) = \delta_A(s, \delta_A(\sigma, q))$.

In the following it is assumed that that plant, specification, and supervisor all have the same alphabet, Σ, and that the controllable and uncontrollable events, Σ_c and Σ_u, respectively, partition Σ.

For an FSM, the set of event sequences possible starting from the initial state defines a formal language.

Definition 2. The *language* of an FSM A is the set

$$L(A) = \{s \in \Sigma_A^* \mid \delta_A(s, i_A) \text{ is defined}\} \quad (1)$$

The interaction between the supervisor and the plant can be modeled by *synchronous composition*, where an event can occur only if it is simultaneously enabled in both.

Definition 3. For two FSMs G and S, their *synchronous composition* is $G||S = \langle Q_G \times Q_S, \Sigma, \langle i_G, i_S \rangle, \delta_{G||S}, M_G \times M_S \rangle$, where

$$\delta_{G||S}(\langle p, q \rangle, q) = \langle \delta_G(\sigma, p), \delta_S(\sigma, q) \rangle \quad (2)$$

[1] This paper will treat neither non-blocking nor minimally restrictiveness, so these properties will not be formally defined.

if both $\delta_G(\sigma, p)$ and $\delta_S(\sigma, q)$ are defined, else undefined.

From the definition it follows directly that $L(G||S) = L(G) \cap L(S)$. Now controllability can be formally defined.

Definition 4. Given a plant G and uncontrollable events Σ_u, a supervisor S is *controllable* w.r.t. to G and Σ_u if

$$L(G||S)\Sigma_u \cap L(G) \subseteq L(G||S). \tag{3}$$

It can be shown that (3) is equivalent to

$$L(S)\Sigma_u \cap L(G) \subseteq L(S). \tag{4}$$

For deterministic FSMs, the language-based controllability definition (4) can be equivalently posed as

$$\forall s \in L(S)\ \Sigma_u(\delta_G(i_G, s)) \subseteq \Sigma_u(\delta(i_S, s)). \tag{5}$$

Here, $\Sigma_u(\cdot)$ represents the uncontrollable events enabled at the state. So (5) says that at a global state reached by the string s, the uncontrollable events enabled by the plant G must be a subset of the uncontrollable events enabled by the supervisor S. In that way, S will never (try to) disable any uncontrollable event enabled by G from any state that S allows the closed-loop system to reach.

Balemi (1992) re-interprets the original R&W SCT formulation into an *input/output interpretation*, where Σ_u are regarded as outputs from the plant and inputs to the supervisor, while Σ_c are outputs from the supervisor and inputs to the plant. This forms a symmetric relation between the plant and the supervisor, see Fig 1 (right), which according to Balemi (1992) is better suitable to model real systems, where events do not occur spontaneously but only as responses to commands. This changes the role of a supervisor from being a passive safety device that stops bad things from happening (while allowing good things), to being an active entity commanding actions of the plant.

It is shown by Balemi (1992) that the input/output interpretation does not really change anything when it comes to synthesis and controllability, the same requirements are still valid and it is only a matter of interpretation. In this interpretation, controllability means that the supervisor must at each global state be ready to accept as inputs the plant outputs enabled in that state. Similarly, there is an *inverse controllability* where the supervisor may not generate outputs (inputs to the plant) that the plant cannot accept in its current state. However, due to the way synthesis is performed, inverse controllability is trivially satisfied, and need not concern us further.

3. INPUT OUTPUT CONFORMANCE (IOCO) TESTING RELATION

After implementing any system, usually testing is carried out to check its correctness. In the case of discrete event systems, the Input-Output Conformance testing proposed by Tretmans (1996) is a way to scrutinize an implementation using specifications as the basis.

In this paper, the point of discussion will revolve around the modified definition of IOCO given by Gregorio-Rodríguez et al. (2013).

Compared to the original definition of IOCO, which considers implementations to be *input enabled*, Gregorio-Rodríguez et al. (2013) consider a simplified version of the definition and applies it to an input-output domain of a labelled transition system. In addition, the original version of IOCO considers *suspension traces*, which are the traces containing quiescent behavior (deadlock). But the modified definition takes all the traces into account, because the quiescent behavior is included in the modified definition by introducing a special symbol for it. Hence, this modified definition of IOCO by Gregorio-Rodríguez et al. (2013) is broader than the original one and will be used throughout this paper.

To give the formal definition of IOCO, consider two disjoint sets of input actions I and output actions O. The output actions are the actions initiated by the system under test and are expressed with an exclamation mark, such as $!a \in O$. Now, we consider a labelled transition system and an example in this section to elaborate the concept of IOCO and give the formal definition.

Definition 5. A *labelled transition system* comprising of inputs and outputs is a 4-tuple $\langle S, s_0, L, \rightarrow \rangle$ where:

- S is a non-empty set of states;
- $s_0 \in S$ is the initial state;
- L is a countable set of labels. These represent observable actions of a system i.e. $L = I \cup O$ where I and O are as above. Consider also a *quiescence* symbol $\delta \notin L$, and define the sets $L_\delta = L \cup \{\delta\}$ and $O_\delta = O \cup \{\delta\}$;
- $\rightarrow \subseteq S \times L_\delta \times S$ is a transition relation such that, $p \xrightarrow{a} q$ implies $\langle p, a, q \rangle \in \rightarrow$ and $p \xrightarrow{a}$ for $a \in L_\delta$, if there exists $q \in S$ such that $p \xrightarrow{a} q$. Similarly, $p \not\xrightarrow{a}$, for $a \in L_\delta$, if there exist no q such that $p \xrightarrow{a} q$. In addition, only coherent quiescent systems are allowed so $\rightarrow$ should also satisfy the following:
  - if $p \xrightarrow{\delta} p'$, then $p = p'$ i.e. a quiescent transition is always reflexive.
  - if $p \not\xrightarrow{!o}$ for any $!o \in O$, then $p \xrightarrow{\delta} p$, i.e. a state with no outputs is quiescent.
  - if there is $!o \in O$ such that, $p \xrightarrow{!o}$, then $p \not\xrightarrow{\delta}$, i.e. no output actions can be performed from a quiescent state.

Furthermore, a trace t is a finite sequence of symbols of L_δ i.e. $t \in L_\delta^*$, including the empty trace ϵ. Additional definitions needed to express the IOCO relation in Definition 9 are as follows.

Definition 6. The set of traces from a state p in an LTS is

$$\mathbf{traces}(p) = \{t \in L_\delta^* \mid p \xrightarrow{t}\}. \tag{6}$$

For an LTS $A = \langle S, s_0, L, \rightarrow \rangle$ its set of traces are the ones defined from its initial state

$$\mathbf{traces}(A) = \mathbf{traces}(s_0). \tag{7}$$

Definition 7. The set of states reached *after* a trace t from a state p in an LTS is

$$p \textbf{ after } t = \{p' \in S \mid p \xrightarrow{t} p'\}. \tag{8}$$

For an LTS $A = \langle S, s_0, L, \rightarrow \rangle$ the set of states reached *after* a trace t is

$$A \textbf{ after } t = \{p' \in S \mid s_0 \xrightarrow{t} p'\}. \tag{9}$$

Definition 8. The set of *outputs* from a state p in an LTS is

$$\mathbf{outs}(p) = \{x \in O_\delta \mid p \xrightarrow{x}\}. \tag{10}$$

The set of *outputs* of a set of states P is

$$\mathbf{outs}(P) = \bigcup_{p \in P} \mathbf{outs}(p) \tag{11}$$

Gregorio-Rodríguez et al. (2013) define IOCO as a relation between states of a single LTS, but an equivalent definition can be made between two different LTSs.

Definition 9. For two LTS, A and B, A is said to be IOCO with respect to B if

$$\forall t \in \mathbf{traces}(B) : \mathbf{outs}(A \textbf{ after } t) \subseteq \mathbf{outs}(B \textbf{ after } t). \tag{12}$$

Here A is the (model of the) implementation, while B is the specification.

The formal definition above describes the IOCO relation such that, an implementation A conforms to a specification B, if and only if for all the traces t in the specification, the **outs** of A after t is a subset of **outs** of specification B after t. From this definition, it can be seen that, for a system to be IOCO, the **traces** in specification is used as a limit to check the implementation, not the other way around.

4. IOCO VS CONTROLLABILITY

In the SCT, controllability is a requirement on the supervisor to never allow the controlled closed-loop system to reach a global state where the plant can generate an uncontrollable event that is not enabled by the specification. If this requirement is fulfilled, the plant cannot uncontrollably take the closed-loop system outside of the specification. If the specification is not controllable, synthesis will in the typical case restrict, or in some cases expand, the specification within the limits set by the plant, to make it controllable and hence usable as a supervisor. Note that there is a duality between the specification and the supervisor, in so far as if the specification is initially already controllable with respect to the plant, then the specification is immediately the supervisor.

IOCO on the other hand, places a requirement on the implementation such that in each global state that the specification and implementation can reach together, the implementation does not enable any output that is not enabled by the specification. If this does not hold, either the implementation is considered faulty and must be altered to become IOCO with respect to the specification, or the specification is considered to restrictive and is altered to make the implementation IOCO with respect to to the specification. Contrary to the SCT, IOCO-testing does not include any techniques to automatically do the necessary alterations.

Comparing expressions (5) and (12), not only are they syntactically similar, but if we regard the implementation as a plant, and the uncontrollable events as outputs from the plant, the two expressions are equivalent.

To further elaborate the equivalence between controllability and IOCO, consider the example of LTSs shown in Fig 2. In the example, we have an implementation G and two different specifications S_1 and S_2. First, we try to establish the IOCO relation using Definition 9. To check the IOCO relation, we first apply the definition to G with respect to S_1 and then with respect to S_2.

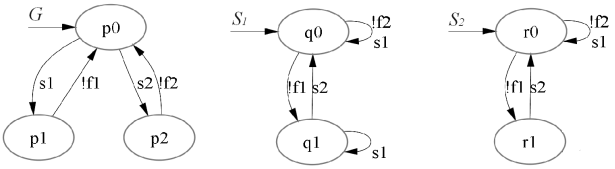

Fig. 2. Implementation/plant G (left), and specifications S_1 (middle) and S_2 (right).

Table 1. IOCO Testing for G and S_1

$\mathbf{traces}(S_1)$	$\mathbf{outs}(G \textbf{ after } t)$	$\mathbf{outs}(S_1 \textbf{ after } t)$
ϵ	$\emptyset$	$\{!f1, !f2\}$
$s1$	$\{!f1\}$	$\{!f1, !f2\}$
$s1.!f1$	$\emptyset$	$\emptyset$
$s1.!f1.s2$	$\{!f2\}$	$\{!f1, !f2\}$
$s1.!f1.s2.!f2$	$\emptyset$	$\{!f1, !f2\}$
$s1.!f1.s1$	$\{!f1\}$	$\emptyset$

According to the IOCO definition, the first step is to analyze all traces in S_1 and G individually and then compute the $\mathbf{outs}(\cdot)$ i.e. possible outputs from the reached state after the execution of each trace.

In the case of G and S_1, we start to evaluate **traces** possible in S_1 from the global initial state $\langle p0, q0 \rangle$. As can be seen from Table 1, the IOCO relation holds for five **traces** only. For the sixth trace in S_1 i.e. the trace $s1.!f1.s1$ Definition 9 does not hold.

To check the failure of the IOCO relation, we analyze the LTS of both S_1 and G to verify the sixth trace mentioned in Table 1. It can be seen that, $\mathbf{outs}(G \textbf{ after } s1.!f1.s1) = \{!f1\}$, while $\mathbf{outs}(S_1 \textbf{ after } s1.!f1.s1) = \emptyset$. Hence, the IOCO relation does *not* hold in this trace because the $\mathbf{outs}(\cdot)$ possible from G is not a subset of specification S_1. Thus, the LTS G is not IOCO with respect to S_1. Now, we analyze the above two LTS i.e G and S_1 to check the property of controllability.

From the perspective of controllability, It can be seen that, the $\mathbf{outs}(\cdot)$ i.e. the uncontrollable events of G are not the subset of $\mathbf{outs}(\cdot)$ of S_1 after the trace $s1.!f1.s1$. Hence, the property of controllability does not hold in case of S_1 and G.

Now, we see that G is not IOCO with respect to S_1 and at the same time S_1 is not controllable with respect to G. Usually in a scenario where the specification is not controllable with respect to G, synthesis is carried out to make the specification controllable with respect to G. The synthesis result is presented in Section 5. Next we will check if G is IOCO with respect to S_2 and at the same time if S_2 is controllable or not with respect to G.

To check the property of controllability and the IOCO testing relation, first we apply the IOCO definition to G with respect to S_2. Again, all traces have to be analyzed in S_2 and G individually and the $\mathbf{outs}(\cdot)$ are compared, i.e the possible outputs from the reached state after the execution of each trace.

In the case of G and S_2, we start to evaluate **traces** possible in S_2 from the global initial state $\langle p0, r0 \rangle$. As can be seen from Table 2, all **traces** mentioned complies with

Table 2. IOCO Testing for G and S_2

traces(S_2)	**outs**(G **after** t)	**outs**(S_2 **after** t)
ϵ	$\emptyset$	$\{!f1, !f2\}$
$s1$	$\{!f1\}$	$\{!f1, !f2\}$
$s1.!f1$	$\emptyset$	$\{!f2\}$
$s1.!f1.s2$	$\{!f2\}$	$\{!f1, !f2\}$
$s1.!f1.s2.!f2$	$\emptyset$	$\{!f1, !f2\}$
$s1.!f1.s1.s2.!f2.s1$	$\{!f1, !f2\}$	$\{!f1, !f2\}$

Definition 9. Hence, the IOCO relation holds because the **outs**$(\cdot)$ possible from G is a subset of specification S_2. Thus, the LTS G is IOCO with respect to S_2.

Here, it can be argued that we have to evaluate *all* the possible **traces** in S_2 to state that the IOCO relation holds. As can be seen from Figure 2, there are infinitely many **traces** possible in S_2. Evaluation of every trace will be an exhausting task, also every time a new trace is evaluated the string gets longer, hence the testing of the IOCO relation becomes tedious.

In both cases the six evaluated **traces** exhaust the possible global state space of G and S_1, and G and S_2, respectively. Hence, ***traces*** *should be evaluated until all possible state combinations have been checked.*

The main reason for the condition mentioned above is, **outs** i.e. the uncontrollable events are evaluated from the state reached after the executed trace for the IOCO relation. Thus, if all possible combinations of states have been explored for IOCO by using a finite number of traces, testing can be called *sound*. Applying the same condition for trace evaluation in case of S_2 and G, it is evident that, after evaluating six traces all possible state combinations have been explored and the IOCO relation holds in every state. Hence, G is IOCO with respect to S_2.

Now to check the property of controllability, that is, whether S_2 is controllable with respect to G, we have to check the **outs** (uncontrollable events) from G and S_2. From Table 2, it is evident that for every trace stated, all uncontrollable events of G are in S_2, hence S_2 is controllable with respect to G.

When altering an implementation that is non-IOCO with respect to a specification, the obvious thing seems to be to restrict the behavior of the implementation so that it becomes IOCO with respect to the specification, in much the same way as to what synthesis does when generating a supervisor. If on the other hand for a non-IOCO system the specification is to be altered, then again this can be done by synthesis but now expanding the specification to make the implementation IOCO. The next section elaborates on these two possibilities.

5. SYNTHESIS

If the specification and the plant do not conform to each other according to controllability or IOCO, there preferably should be some algorithmic way to remedy the situation. Within the SCT, this is called *synthesis*, and it creates a supervisor that is by construction controllable with respect to the plant. There are two basic ways to do synthesis given a plant G and a specification K, either to calculate the *supremal controllable sublanguage*, denoted $\mathbf{sup\,C}(G||K)$, or calculating the *infimal controllable superlanguage*, denoted $\mathbf{inf\,C}(G||K)$. It is known (Ramadge and Wonham, 1987; Cassandras and Lafortune, 2009) that both of those languages always exist, are computable (in the regular case), and are unique; however, $\mathbf{sup\,C}(G||K)$ may be the empty language (signifying that no useful solution exists) while $\mathbf{inf\,C}(G||K)$ may be the entire plant.

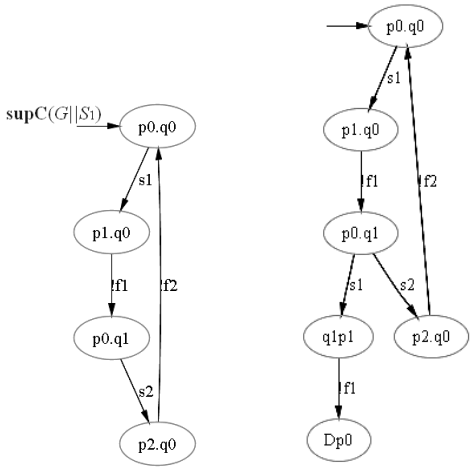

Fig. 3. Synthesis result for G and S_1. Left, supremal controllable sublanguage. Right, infimal controllable superlanguage.

The result of calculating the supremal controllable sublanguage of S_1 with respect to G, using the tool *Supremica* (Malik et al., 2017), is shown in Fig. 3 left, denoted by $\mathbf{sup\,C}(G||S_1)$. Let us call this automaton S, and it is such that $L(S) = \mathbf{sup\,C}(G||S_1) \subseteq L(G) \bigcap L(S_1)$. S would be a supervisor able to control the plant G to always remain within the specification K. However, in a testing setting we can use S as a replacement for G, which is not IOCO with respect to S_1. This can be done since S_1 is controllable with respect to S, and hence S is IOCO with respect to S_1, as is straightforwardly proven. So, in place of the originally given implementation G (Fig 2), which allows arbitrary sequences of $s1.!f1$ and $s2.!f2$ loops, the restricted implementation $\mathbf{sup\,C}(G||S_1)$ of Fig 3, which only allows $s1.!f1.s2.!f2$ loops, can be used, and this is IOCO with respect to S_1. In practice, S could be given as feedback to a developer and used as specification for what should be implemented, guiding the developer in correcting the implementation.

However, in some cases restricting the implementation to be IOCO with respect to the specification may not be a good alternative. One such case would be when $\mathbf{sup\,C}(\cdot)$ is empty. Then, instead of restricting the implementation, the specification can be enlarged. This is done by calculating the infimal controllable superlanguage. For S_1 and G, the result of this calculation, $\mathbf{inf\,C}(G||S_1)$, using the tool DESUMA (Ricker et al., 2006), is shown in Fig. 3, right. In this case the original implementation G is now IOCO with respect to the enlarged specification $\mathbf{inf\,C}(G||S_1)$, or equivalently, the enlarged specification $\mathbf{inf\,C}(G||S_1)$ is controllable with respect to the original plant G. In practice, $\mathbf{inf\,C}(G||S_1)$ could be given as feedback to a developer help understanding what needs to be relaxed for the implementation to be valid.

In the development of a safety-critical system, restricting the implementation to the supremal controllable sublanguage would seem to be most appropriate, as this guarantees that the implementation never breaks the specification. However, for large systems it is not easy to assess whether important parts of the implementation behavior have been removed by the synthesis, the only guarantee is that the restricted implementation is safe with respect to the specification, not that all of the specification is achieved. On the other hand, enlarging the specification to make the given implementation IOCO, adds behavior from the implementation to the specification, and it might not be easily assessed exactly what behavior has been added, other than that this is behavior that breaks the original specification. So, both alternatives of algorithmic remedy for a non-IOCO system seem appropriate in a testing setting, but the particular choice seems very application dependent.

6. CONCLUSIONS

The relationship between the controllability property from the SCT defined to guarantee well-behaved supervisory control, and the IOCO property used for conformance testing of an implementation relative a specification, has been examined. It is shown that under the interpretation of the uncontrollable events as outputs from the plant, and viewing the implementation as the plant, the two properties are equivalent. From a controllability perspective the supremal controllable sublanguage of the specification seems to make the most sense, as this guarantees that the controlled system always remains within the specification; only in the case where the resulting supervisor is too restrictive, such as the degenerate null supervisor, would the infimal controllable superlanguage seem appropriate. However, controllability poses a requirement on the specification, whereas IOCO poses a requirement on the implementation, and so correcting a non-conforming system could more reasonably be done by either the supremal controllable sublanguge, or the infimal controllable superlanguge. In the first case the implementation would be restricted to be IOCO with respect to the specification, while in the latter case the specification would be extended to make the implementation IOCO.

There exist efficient methods to asses whether a system of plants and specifications are controllable, such as the *modular* approaches (de Queiroz and Cury, 2000; Åkesson et al., 2002). The equivalence between controllability and IOCO means that those methods can also be used to efficiently assess whether a system consisting of implementation and specification components is IOCO. This idea is investigated by Van Der Bijl et al. (2003), where it is shown that the IOCO relation is suitable (under some restrictions) for compositional testing, but no algorithm to assess whether a modular system is IOCO is given.

For non-deterministic systems, neither controllability nor the IOCO relation are detailed enough to capture the necessary relations between the plant/implementation and the specification. Several generalizations have been proposed, among them the process theoretic notions of *partial bisimulation* (Baeten et al., 2011) for controllability, and the *input-output conformance simulation* (Gregorio-Rodríguez et al., 2013). In future work we will examine the relation between these two properties.

REFERENCES

Åkesson, K., Flordal, H., and Fabian, M. (2002). Exploiting modularity for synthesis and verification of supervisors. *IFAC Proceedings Volumes*, 35(1), 175 – 180. 15th IFAC World Congress.

Baeten, J., Van Beek, D., Luttik, B., Markovski, J., and Rooda, J. (2011). A process-theoretic approach to supervisory control theory. In *American Control Conference (ACC), 2011*, 4496–4501. IEEE.

Balemi, S. (1992). *Control of discrete event systems: theory and application*. Ph.D. thesis, Swiss Federal Institute of Technology, Zürich, Switzerland.

Cassandras, C. and Lafortune, S. (2009). *Introduction to Discrete Event Systems*. SpringerLink Engineering. Springer US.

de Queiroz, M.H. and Cury, J.E.R. (2000). Modular supervisory control of large scale discrete event systems. In S.G. Boel R. (ed.), *Discrete Event Systems: Analysis and Control*, 103–110. Springer US, Boston, MA.

Gregorio-Rodríguez, C., Llana, L., and Martínez-Torres, R. (2013). Input-output conformance simulation (iocos) for model based testing. In *Formal Techniques for Distributed Systems*, 114–129. Springer.

Malik, R., Åkesson, K., Flordal, H., and Fabian, M. (2017). Supremica-an efficient tool for large-scale discrete event systems. In *IFAC World Congress, Tolouse, France*.

Ramadge, P.J. and Wonham, W.M. (1987). Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1), 206–230. doi:10.1137/0325013.

Ricker, L., Lafortune, S., and Genc, S. (2006). DESUMA: A tool integrating GIDDES and UMDES. In *2006 8th International Workshop on Discrete Event Systems*, 392–393. doi:10.1109/WODES.2006.382402.

Tretmans, G. (1996). Test generation with inputs, outputs and repetitive quiescence, 1996. *URL http://doc. utwente. nl/65463*, 46.

Utting, M. and Legeard, B. (2007). *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Van Der Bijl, M., Rensink, A., and Tretmans, J. (2003). Compositional testing with IOCO. In *Formal Approaches to Software Testing*, volume LNCS 2931, 86–100. Springer.